AGENT-BASED FRAMEWORK FOR DISCRETE ENTITY SIMULATIONS[1]

Mark Cowan
U.S. Army Engineer Research and Development Center
Vicksburg, MS 39180

ABSTRACT

Agent-based modeling represents a new way of simulating the interaction of objects with their environment and among themselves through communication. Agent technology incorporates many of the features of more traditional (Lagrangian and Eulerian) efforts but has become feasible in modeling more complex systems only recently. This paper describes the development of a general parallel agent-based modeling framework in C++ on Department of Defense High Performance Computing (HPC) machines at the U.S. Army Engineer Research and Development Center (ERDC) Information Technology Laboratory (ITL) in Vicksburg, MS. It provides background on the motivation behind agent-based modeling and how it extends traditional modeling techniques. Differences are identified, and the strengths and weaknesses of various modeling paradigms are explained. A short history of continuum and discrete model coupling is provided, followed by a description of how agent-based techniques can incorporate features of both Eulerian and Lagrangian models. The architecture of the ITL agent framework and the construction of the behavioral functions that excite or inhibit agent behavior are presented in detail. The hardware/software evolution path is described as the code goes from a small, single-threaded binary running on a Linux workstation utilizing database calls (to meet memory requirements) up to its successful translation as a parallel implementation on large HPC machines. Porting and scaling difficulties are fully explained.

The framework is tested on an idealized ecological sandbox representing the Noyo River basin, California, and the virtual growth of submerged aquatic vegetation (SAVs) under hydraulic conditions driven by the ADH (ADaptive Hydrology) code output of the ERDC Coastal and Hydraulics Laboratory. Designed to be run as a general simulation test-bed across many sciences (not *exclusively* ecology), the framework easily allows modification of the agent behavioral functions. The emergence of collective behaviors among the agents can be traced, and the framework permits researchers to roll back the history of events described by the model to determine how large-scale patterns may have been initiated. The agent-based model, of course, emulates only a small portion of what is occurring in the real-world ecology. These aspects are discussed, and fundamental questions are raised regarding model validation versus real-world experimentation. Agent-based modeling is advanced as an ideal test-bed for researchers testing complex dynamic system interactions and as a powerful method to “prune out” weaker scientific hypotheses. The limitations of the general framework are delineated, and a future work path is sketched out that could reduce or eliminate the current weaknesses in the system.

1. DEFINITION OF AN AGENT

As a measure of a tool's maturity within the computational community, wide variability in definition serves as a precaution. Agent-based modeling (ABM), along with many other techniques falling under the aegis of artificial intelligence, has unfortunately become a victim of its own propensity to overpromise initially and undersupply in the long run. For those outsiders to the community who are compelled to ask, "What is a computational agent?", the answer is at present highly dependent upon to whom the question is directed. Some authors list eight or nine essential features, ranging from the mundane and easily accomplished to fantastic visions on the cutting edge of artificial intelligence research. This article will err on the side of simplicity. Following Russell and Norvig’s lead in their *Artificial Intelligence: A Modern Approach* (Russell and Norvig 2003), this article describes an agent as a computational entity embedded within an environment having sensors to recognize its current state and effectors to act upon its environment (Fig. 1). In effect, a computational agent serves as both a witness to and an actor upon its environment. This approach avoids many of the serious philosophical problems of anthropocentric descriptions, while being broadly inclusive of much recent solid computational agent work.

[1] Approved for public release; distribution is unlimited.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **01 NOV 2006**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Agent-Based Framework For Discrete Entity Simulations1**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **U.S. Army Engineer Research and Development Center Vicksburg, MS 39180**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

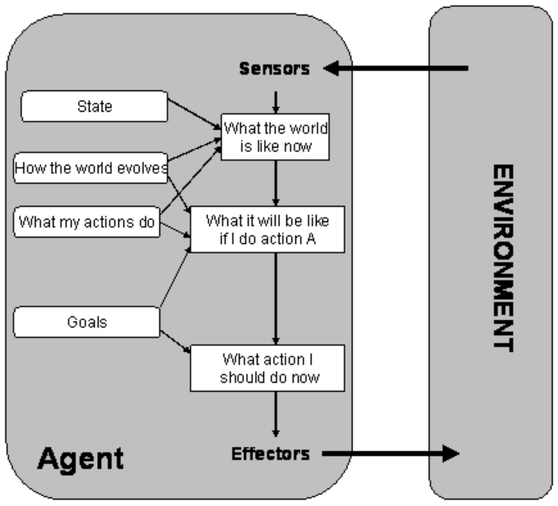
12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002075., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**8**	


Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Effectors
Agent
ENVIRONMENT

.

Fig. 1. An agent conceptual schema (from Russell and Norvig, 2003, p. 50. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, New Jersey)

While an agent is a software entity situated in a world, it should be noted that much of that world is probably outside its sensory range and hence contributes to uncertainty. An agent reacts to components within its neighborhood, within its sensory envelope. There it can determine its current status (i.e., whatever property is explicitly of concern) and using effectors--physical, computational, or some combination of the two--can act upon that world to effect change, ideally improving its own status. Outside of its sensory range, the agent is oblivious to what is occurring, although effects could be transmitted indirectly into its range by neighboring agents that can sense the changes.

Note what is missing from this description of a computational agent: there is no supposition of memory of previous attempts at changing the world and hence the agent's status. The absence of memory explicitly denies any chance of improvement in efficiency of actions taken. Similar environmental conditions and agent reactions to those conditions could recur without the agent recognizing its repetition, much as looping behavior is possible (but not assured) in Conway's Game of Life (Gardner, 1970). Usually to reduce these repetitions and to grant the agent the ability to improve over time, a memory--though finite in extent--is permitted the agent to allow for self-driven behavior modifications that may increase the probability of achieving its goals. This, in fact, is a widening of the concept of sensor, since it implies an interiorizing of its sensory range: the self-recognition of the agent's previous and current statuses, the role its previous decisions had in the transitions from the past to the present, and a way of retaining that causal link for similar cases in the future.

No preconditions are placed upon the world--the world simply *is* and the agent interfaces with and interprets that world under conditions of uncertainty by reading changes within its own sensory range. Other aspects of the world away from the agent, it is assumed, could be vastly different. Ideally, the world should be represented as an adaptive, unstructured mesh whose form is dictated by the underlying continuum mechanics. Obviously the simpler and more static the computational mesh, the less computationally intensive the agent modeling effort will be. However, for generality, the ideal agent framework should encompass the computationally difficult cases.

More demanding definitions of agents presuppose interagent communication devices with memory, which could promote the development of complex cooperation, competition, and parasitic strategies. However, if, as in the present paper, the only communication taking place in the agent-based model is indirect, through one agent's effect upon its environment and another's recognition of the change through its querying of the world, then the agents may be accurately described as reactive, i.e., they do not act upon one another directly, but rather only upon their local environment from which all neighboring agents must draw.

What role in computational simulations do the proponents of agent-based modeling seek? Do they wish to usurp the positions of established methods, boldly staking claim deep within the simulation frontier where few have dared to go, or to hybridize diverse, known techniques allowing easier manageability? This author believes that the latter is the case, and that in concert with parallel processing techniques, ABM can foster the successful marriage of Lagrangian and Eulerian models. Where Lagrangian models treat the individual particles as frames of reference and Eulerian models take a God's-eye view over the whole of the system under consideration, agent-based modeling attempts a commingling of the two---focusing upon agents as particles, each distinct and obeying potentially unique behavioral rules, intimately embedded within an environment driven by continuum mechanics. Agent-based modeling ideally takes the best of both paradigms--the atomistic perspective of individual agents (demanded by Lagrangian techniques) within the overall environment (ruled by Eulerian computational meshes).

2. HISTORY OF CONTINUUM-DISCRETE MODEL COUPLING

The required processing power is just now becoming available to contemplate attacking problems as challenging as those addressed by agent-based modeling, and valiant attempts have been made recently to marry the best of these two distinctly different paradigms short of building an agent application. The resulting tools, however, are usually focused upon very specific scientific problems, and the research requirements in these cases seemingly lack any impetus to develop general tools for simulationists across the wide range of scientific disciplines. Much of the difficulty here lies in the question of how best to design an interface for the numerous continuum models that underlie the agent-based modeling effort. Obviously, many of those continuum models are emulating wildly different physics, divergent in scope, input, and expected output. To expect a tool to interface smoothly without hindrance across the whole family of pre-existing continuum models that could serve as drivers for agent-based modeling is simply to set the bar too high. Even in those cases where similar (but restricted) frameworks have been attempted, e.g., the Swarm application (Hiebeler, 1994), several key ingredients for generality have been lacking: multiple spatial dimensions, widely used language for development, portability to mainframe systems to profit from scalability implicitly suggested by agent paradigm, and others.

3. GOAL OF AGENT-BASED MODELING (ABM)

For any generally biological entity, a simulationist must define its behavioral/life-cycle aspects for modeling as a computational agent. The agents are then instantiated and placed in a user-defined manner into the computational (preferably, mixed element and adaptive) grid. Continuum model calculations are made that provide numerical descriptions of the mechanics under consideration for each node in the mesh for each time-step in the history of the run. Per time-step each agent reacts to the changes occurring at its neighboring nodes. Changes in the agents potentially affect each other (locally) and ideally feed back into the continuum model, affecting the physics in the next time-step for its locale (if its impact is decisive enough to be a driving factor in the continuum model's future time-steps). To benefit from the parallel processing architecture for which agent-based modeling is so well-suited, all agent runs are made across multiple CPUs, while synchronizing the results between continuum model time-steps. It should be noted that the continuum model drives the clock, and accordingly the agent's behavioral functions must be so designed to account for the continuum model's temporal resolution. To account for essential mobility characteristics of fauna in ecological models (soldiers in a combat zone, pedestrians in an urban landscape, or any other moving biological entity), particle tracking of the individual agents in the computational grid will be required. Also as communication can occur across more complex biological communities, a communication model allowing local interaction among entities may be desired by some simulationists. The latter feature could lead to investigations of community cooperation, competition, prioritization of life necessities, and with the introduction of extensive memories, the eventual development of strategies to play to an individual's strengths relative to its community to maximize efficiencies.

4. PREFERENCE FOR ITL’S GENERAL AGENT FRAMEWORK

The behavioral functions that drive the agent life-cycle are easily extensible. They are simple math functions, and typical interactions (such as function composition) are very simple to code using the pre-existing mathematical libraries. As would be expected, the mathematical domain (i.e., the abscissa) driving the functions may be spatial, temporal, or other.

As designed, the architecture is flexible and accommodating for various modeling paradigms. Its handling of all parallelization and message passing issues permits the model developer to focus solely on agent behavior and rules for interaction with environment. More precisely, all parallel processing libraries are built into the general framework engine; due to DBuilder and its calls to the message-passing interface (MPI) libraries, there is no need to adjust or "roll your own" parallelization routines.

The agent framework, as currently coded, requires no proprietary code. It evolved from open-source development tools onto the ERDC Major Shared Resource Center (MSRC) computers; and although it uses DBuilder/MPI, all needed calls are ANSI-compliant and could just as easily have interfaced with the MPIch freeware.

By their autonomous nature and limited sensory regions, agents readily demand a parallel layout. Fortunately for developers, farming out the agents across multiple CPUs offers fast turnaround on even staggeringly large runs. It follows that debugging behavioral functions and interactions is relatively simple since the developers have the opportunity to catch errors and rerun quickly.

The format of the run output can be designed to leverage current visualization tools to aid in this debugging. However, slight adjustments ("tweaking") of current visualization tools are possible to investigate more complex behavior than that for which the tools are currently coded.

A base class for agents (coded in C++) is currently provided to developers who wish to add additional layers of complexity to their agents simply by using inheritance. Again, the fundamentals of the agents (namely, a unique identifier and spatial coordinates) that are necessary for the framework to do its job are hidden from the developer, while these fundamentals are inherited and composed with the behavioral and life-cycle components of the agents that give them their unique responses to the environment they find themselves situated within.

5. THE EVOLUTION OF ITL'S GENERAL AGENT FRAMEWORK

Hitherto, the ideal general agent framework has been described. As currently coded, the framework is lacking in some key characteristics, with the divergence from the ideal described in the following text.

Originally, the SAV demonstration was viewed as a "proof of concept" exercise. Obviously limited in scope as a general modeling tool, if successful, it would show the vague outlines of what was computationally feasible using simple agents over multiple processors.

As a first step, an SAV demo was prototyped in Perl on single Intel Pentium 4 CPU running Red Hat Linux 9.0 and using a Postgres database server for environment queries of neighbors and continuum data. As expected for raw database queries (no database optimizations in place), the tests were generally slow,; but overall the results were positive, and the code was debugged through several testing iterations in this form.

Eventually the code was ported to GNU C++ on the same single Intel Pentium 4 CPU running RedHat Linux 9.0 and Postgres database server. Again Postgres was used for environmental queries, and the tool remained relatively slow because of the immense number of queries necessary to assess the environmental condition of each agent and respond accordingly per time-step.

Once the code had been demonstrated to be amply "bulletproof" on the single CPU, it was time to move the C++ code to the MSRC HPC computers and test it there on a single CPU, again using a Postgres database server (same purposes as above). This was just an exercise in correctly transferring the code and handling any warnings or error messages resulting from the compiling of the code upon a different compiler. Although the Postgres database was obviously on a different server than before, the installation of Postgres proceeded along the lines of the Red Hat Linux installation (using the same notes, developer permissions, and table schemas), so the results were painless.

Because the DBuilder/MPI code was more amenable to a different MSRC HPC computing environment, it was decided that the C++ code should be moved. The compile step was similar enough so that few problems were created for the porting effort. At this stage, all Postgres database queries were dropped because of concurrent changes in MSRC HPC Kerberos security policy (namely, a considerably shortened Kerberos ticket duration). Certainly, Postgres allowed for a massive simplification in the development code since SQL provided a natural and easy query device into the mesh topology. However, it was the primary bottleneck to quick processing of environmental queries, and it was known from the earliest stages of development that Postgres was simply a placeholder for the larger effort of integrating in the DBuilder code. As a wrapper to the MPI calls, DBuilder assists in optimizing farming out work to multiple CPUs and maintaining information regarding network topology and how nodal neighbors are mapped in the computational grid. This effort was complemented by a full suite of functionality testing and debugging, followed by a demonstration of results to the ERDC management staff and technical leadership.

Modeling the SAVs successfully over multiple CPUs led to a new requirement that the underlying modeling engine be generalized, to lose the "SAV only" bias and to move toward a framework more open for other uses. The SAV-specific Species class was moved out, and only the unique agent ID number and the three-dimensional spatial coordinates were retained in the base class to allow inheritance by other more specific agents. These remaining features were deemed sufficient for any agent embodied within three-dimensional space; other features could be added on by the simulationists as needed. Also the SAV-specific code that has resided in the main function of the engine was moved into a process function that is user-modifiable. Of course, the Makefile was updated to address the breaking-out of this functionality into separate object files.

6. HARDWARE AND SOFTWARE CONSIDERATIONS

The general agent framework is written in ANSI-compliant C++ code that is dependent upon the DBuilder wrapper to MPI functions. As stated previously, the SGI MPI code that DBuilder calls is perfectly compatible with the open source MPIch. Our framework runs an MSRC HPC cluster, which utilizes the SGI O3K Irix operating system, version 6.5.19. Both gmake and hand-made Makefiles are used to build the necessary executables.

As performance is highly architecture dependent, a description of the cluster is necessary. It comprises SGI O3K machines, running 512 400-Mhz CPUs and 1024 700-Mhz CPUs, having a total computational capacity of 1,844 Gflops and an aggregate memory of 1.536 TB. The network interface is gigabit ethernet, and the total disk storage is 20 TB, connected with fibre channel RAID5. As mentioned above, the machines run SGI O3K Irix, version 6.5.19, and SGI MPI, version 1.7. The average load is estimated at 70-80 percent capacity during most working hours.

7. SUBMERGED AQUATIC VEGETATION (SAV) DEMONSTRATIONS

In mid-January 2004, we received from Ms. Jackie Hallberg of USACE ERDC Coastal and Hydraulics Laboratory (CHL) (personal communication) a couple of ADH model runs for the Noyo River near Fort Bragg, CA (Fig. 2). CHL provided the model output for 24 time-steps and the complete computational mesh (all triangular elements) for the Noyo including a proposed breakwater structure. The Noyo River feeds into the Pacific Ocean south of Fort Bragg, CA, between Soldier Point and Todd Point. This particular area is interesting hydraulically in that it possesses, in a relatively small area, normal riverine characteristics in addition to having tidal effects from the ocean. These features are both present in the ADH output provided by CHL. Although the hydraulic obstruction does not exist in the real world, it was retained in the geometry since it slows the water velocity and generally promotes the growth of submerged aquatic vegetation on the seaward side. Such potential future construction scenarios are often part of U.S. Army Corps of Engineers design projects, which are compared for hydraulic, economic, and environmental factors (Fig. 2).

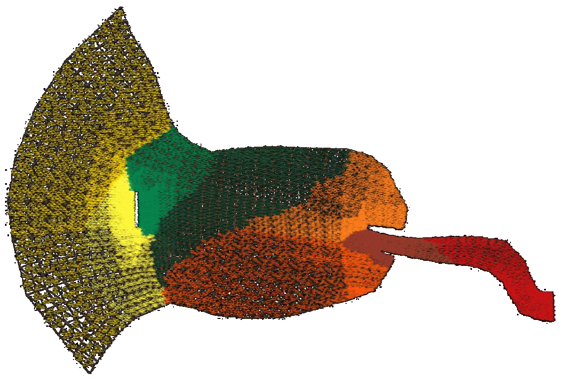


Fig. 2. Noyo River basin with hydraulic obstruction.

The SAV demo executable brought the computational grid into memory using DBuilder's hooks into the MPI. With DBuilder we have direct insight into which vertices compose an element, which vertex lies upon a particular CPU, and other essential information regarding mesh topology. Each nodal point in the mesh was given a unique gridpt ID, and all velocities calculated from the hydraulics models were normalized into speeds since direction of flow has no effect upon the growth cycles of aquatic plants, although speed does. Across all vertices in the mesh, the plant nutrients were instantiated to a particular level conducive to plant growth (as determined by the growth functions for the agents). Note that the descriptor "nutrient" is one of several fields in the WorldGridPt object for use in tracking features of the environment that are not handled by DBuilder, and it is solely SAV-related, i.e., not an element of the general framework. In the agent code, a species is established, having developer-defined life-cycle features, including biomass size at birth, maximum growth rate as a function of age, optimum nutrient levels, reproduction age range, maximum and minimum agent density for reproduction, maximum and minimum water speed---in short, all of the conditions under which an agent can live and potentially, thrive. Over the 24 time-steps of the continuum model runs, the agent determines its nearest WorldGridPt node, requests its needed nutrients (if available), adjust its own biomass to compensate for growth or decay, and passes the required information to adjust the nutrient level at that WorldGridPt. The code cycles through all agents across all CPUs and waits until all are ready for next time-step (using MPI_Wait within DBuilder), then it goes to the next time-step until all time-steps have been worked through.

8. DATA RETAINED FROM DEMONSTRATION RUNS

The data retained from the modeling runs are broken into two categories, both reflective of the objects involved: (1) agents and (2) world grid points. For the agents, the following information is stored in ASCII logs for later analysis: unique agent identifier; Euclidean x,y,z positions; biomass; current age; and model time-step. The data for the world grid points are simpler, since the agents are only doing a nutrient exchange: unique world grid point identifier, model time-step, and nutrient. (Of course, for other modeling purposes, any number of properties such as nutrient could be retained here.) A post-mortem analysis of the model run can be extremely useful for checking that the agent behavior meets expectations, for debugging code anomalies, and for acceptable runs, serving as raw data for visualization applications. Intrinsic to using MPI is keeping tabs on the CPU from which any information is derived. To this end, all logs retain a rank value, i.e., a unique CPU identifier, that precedes all other logged output.

Seasoned programmers know the benefit of logged output of runs. However, at different stages in the developmental iterations, more logged output is desired than at others. The ability to change log levels of output from a compiler directive radically simplifies the debugging process and permits scaling back to a level of minimal (or no) logging once the code is operational. To assist in staggering the log levels, some open source log level code was obtained online and modified to fit our needs. This code allows the developer to freely increase or decrease the granularity of details in the log output from a demo run. Under this scenario, the log level is set prior to the code build; and, since the same binary is run across all CPU under the MPI libraries, a uniform level of logs would report back from all CPUs to a central ASCII log.

9. USER-DEFINED BEHAVIORAL FUNCTIONS

The behavior of agents is driven by user-defined curves, very similar in form to those driving Monte Carlo simulations. These mathematical functions can be simplified down to constant (uniform) values or be composed together to attain much more complex effects. Functions currently coded as part of the framework's toolbox include triangle, increasing linear sigmoid, decreasing linear sigmoid, increasing (Heaviside) step, and decreasing (Heaviside) step functions.

10. ECOLOGICAL CONSIDERATIONS

In any modeling effort, the model development and the science must find a common meeting ground, the position at which the emulation of the physical phenomena is "good enough" for the purposes of model construction. To overstep these bounds in an attempt to gain more model validity would be economically wasteful. To undershoot the goal would be unethical and potentially dangerous in that the model would not reflect the minimum information necessary for decision making and may, as a consequence, lead to life-threatening situations.

That being said, the growth parameters and behavioral ranges in the SAV demo runs were simply chosen for model stability and some resemblance to biological behavior under similar environmental circumstances. There was no desire to emulate behavior of any particular botanical species, and, as such, validation of model results to empirical observations was not sought. All SAV demos were just a proof of concept for agent-to-agent and agent-to-world interaction.

Suppose, however, that eventually we wish to validate this model by contrasting its output per time-step with the results from a corresponding empirical study. Several essential questions, which we do not pretend to answer here, arise: How do you know when you have reached a steady-state condition in the agent model, especially as these models are generally nonlinear? Many simulationists have undoubtedly had to face down the "engineering judgment" evasion, but besides that, a very real question remains: Does any numerical criterion exist that can highlight the steady-state boundary?

Related to this is a fundamental question regarding how to set up the a priori ecological conditions. As run, the SAV demo assumed a virgin environment prior to the first time-step; i.e., ecologically, the primary succession stage was taken as the given (although it is fairly rare in nature). Usually one would expect a secondary succession condition having competition between intruder and indigenous populations over quite some time. So, how to formulate this secondary succession condition prior to the first time step of an agent model run? Would not all of the behavioral functions for all indigenous species be required as well as the seed conditions that drove the nonlinear system to this stage? Obviously, this leads to an undesirable infinite regression computationally, and hence, the question: how to establish a functional, stable, credible environment in which to introduce the SAVs?

To emphasize, how does the simulationist code behavior into an individual agent to induce realistic, emergent behavior within its community? How does one initialize/seed the a priori environment to emulate (to an acceptable approximation) the tensions within the ecosystem before the introduction of the SAVs? What is the seeding strategy for introducing the SAVs into the environment; e.g., individual plantings uniformly throughout the environment, plant spread by runner behavior and the like, etc.?

These seem to be the fundamental questions of how to model (here, SAV) agents to achieve realistic environmental dynamics that can be validated to an acceptable degree of accuracy. (And ones upon which this research has not focused. The general agent framework, however, can provide a rich test-bed for various environmental scenarios, which may lead to a significant "pruning" of the space of possibilities arising from these questions).

11. CURRENT LIMITATIONS TO THE GENERAL AGENT FRAMEWORK

There are several limitations to the general agent framework as it is currently coded.

First, it is assumed that the computational mesh is composed exclusively of triangular elements. The scope of the framework's architecture must be widened to handle the initializations of agents into mixed element meshes. Since DBuilder is so intimately connected with all agent interactions with the grid, it is fortunate that DBuilder, as currently coded, will handle all topology queries regardless of the geometry of the mesh elements. As a result, the changes to the framework are fairly minor. (Note that although DBuilder was designed to handle topological queries across both structured and unstructured [i.e., mixed] meshes, it is relatively untested for the latter case; so it may require further testing to ensure correct results).

Secondly, it is assumed that the primary environmental properties (e.g., nutrients for the SAV demo) are uniformly distributed across all mesh nodes at the first time-step. This assumption is quite unjustified empirically, although for validation, determining the proper distribution would require copious field data. Certainly the framework should be extended to permit a more realistic spatial distribution. As it currently stands, the resolution of the mesh drives the distribution of these nutrients. More acceptable would be an algorithm that would disregard the dictates of the mesh triangulation and focus instead upon Euclidean coordinates. In general, the distribution of nutrients (or any other environmental entity) should be unhindered no matter what scheme is finally chosen by the user. A complete solution to this limitation would involve placing the nutrients where desired and forcing the framework to interpolate from the given positions to the known mesh nodes, which ultimately drive the computational runs.

Thirdly, as a holdover from the SAV demo, an agent is initialized into its triangular element along a randomly chosen median of the triangle, with the local number of agents per triangle predefined by the user. This is a fairly severe limitation. Ideally, the agents should be able to occupy any spatial position within the confines of the grid. The scheme as chosen was an effort to simplify the geometry algorithms, to prevent MPI issues with shadow nodes, and to give enough distance between agents so they would not be smothered out early on by overdensity in the SAV demo runs. Similarly, note that if reproduction occurs, the child occupies the same spatial coordinates as the parent--this restriction was an effort to prevent a parent residing on one CPU from spawning a child onto another CPU. Basically, all calculations regarding the reproduction resided on the same CPU as the parent; it was cleaner that way. (Actually if one assumes a shared memory architecture, it is an irrelevant consideration; however, without this restriction, one can run into some issues for a distributed memory architecture.) Choosing to place agents along the median of the triangle was one of many equally "easy-to-code-and-test" possibilities. With the introduction of mixed meshes (as described above), other techniques for placing the agents into the mesh will be required. Overcoming this restriction is a case of choosing an algorithm or set of algorithms that permit simulationists to seed as they want, rather than be limited by the framework to predefined choices. The framework, at present, allows the changing out of this portion of the code by the model developer as it is a part of the external library of functions (not the engine), although at present only one example is in place.

Fourthly, we have a significant issue that could force a fork in the code, with all of the software engineering process hassles that entails. As described above, an agent queries through DBuilder to determine its nearest mesh node and bases its behavioral choices for this time-step upon the nutrients available there (answering, how much food?) and the number of agents residing in its own (triangular) element (answering, how many near me?). The agent extracts its required nutrients from the node based upon the nutrient value there at the beginning of the time-step. However, if the node is farmed out to another CPU (than the one the agent resides on), then knowledge of nutrient extraction by other agents is gained ONLY at the next time-step. Hence, the rare case may exist when an agent extracts more nutrients from a node than are available after others have done their own extractions. So, how can this issue be resolved? One cannot in a parallel environment simply update the nutrients upon any extraction and require all affected nodes to read (or re-read) the available nutrients--the computational overhead is far too high. A shared memory architecture offers a fairly low overhead solution by the creation of an globally shared array for each node. Before the agent makes an extraction, it must query the available nutrients against that globally maintained array. This is a clean solution, but it works only for this particular architecture. For distributed memory systems, a globally maintained array is not a possibility; one can assume generally that each CPU handles its own memory, the contents of which may become out-of-date and hence unrepresentative of the current state of nutrients. One could mark those rare cases where an agent would be reacting to nutrients in a ghost node (i.e., a node whose existence is known locally, but whose contents are governed by another CPU). Upon completion of the time-step, those few agents' extracted nutrients could be compared to what was in fact available at the time. If their requirements were in bounds, no further action need be taken; however, if their needs exceeded what was in fact available (due to extraction by other agents), then one could roll back the effects to keep the nutrient extraction level within bounds. Due to the nonlinear nature of agent modeling, this algorithm could have far-reaching consequences and result in a large computational burden as agents advance by assumption and retreat as reality sets in. At present, no other technique to resolve this issue is known. Obviously this direction should be avoided if at all possible. Since no one would wish to run the distributed memory algorithm unless absolutely necessary, this would force a code fork, where two distinct paths were designed, coded, tested, and maintained corresponding to the two architectures, namely, shared versus distributed memory systems.

Fifthly, resulting from using SAVs as the driving mechanism for the development of this agent framework, agents are currently immobile (i.e., they are not using particle tracking). Dr. Andrew Goodwin of the ERDC Environmental Laboratory Columbia River Basin Research Facility, North Bonneville, WA, (personal communication),has recently provided the Fortran interface he has implemented for initializing and calling the particle tracking library of Dr. Ruth Cheng, MSRC. Further analysis of his code design and rewriting aspects of it in C++ are required before particle tracking will be fully realized in the general agent framework. To drive the testing scenarios, the SAV demo will be slightly modified to allow for agent motion per the CHL velocity calculations. The most difficult part would be verification of model consistency by tracking several agents over time and searching for anomalous movement.

ACKNOWLEDGMENTS

My thanks to Dr. Charles R. Welch, Mr. Chris Merrill, Dr. Cary Butler, Mr. David Richards, Dr. Jeffery Holland, Dr. John Nestler, Mr. Bobby Hunter, Dr. Ruth Cheng, Mr. Barry White, Dr. Stacy Howington, Ms. Jackie Hallberg, Mr. Richard Walters, and Mr. Paul Adams, all of ERDC, Vicksburg, MS.

REFERENCES

Gardner, M., 1970: Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game 'Life,' *Scientific American*, Oct. 1970, 120-123.

Hiebeler, D. E., 1994: The Swarm Simulation System and Individual-Based Modeling, presented at and published in the proceedings of *Decision Support 2001: Advanced Technology for Natural Resource Management*, Toronto, September 1994. Also published as Santa Fe Institute working paper 94-12-065.

Russell, S. J., and P. Norvig, 2003: *Artificial Intelligence: A Modern Approach*. 2/E, Pearson Education, Inc., Upper Saddle River, NJ, 1132 pp.